JSContact: A JSON representation of contact data

Abstract

This specification defines a data model and JSON representation of contact card information that can be used for data storage and exchange in address book or directory applications. It aims to be an alternative to the vCard data format and to be unambiguous, extendable and simple to process. In contrast to the JSON-based jCard format, it is not a direct mapping from the vCard data model and expands semantics where appropriate.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 May 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in
Table of Contents

1. Introduction
   1.1. Motivation and Relation to vCard, jCard and xCard
   1.2. Notational Conventions
   1.3. Data Type Notations
      1.3.1. Objects and Properties
      1.3.2. Type Signatures
      1.3.3. Property Attributes
      1.3.4. The @type Property
   1.4. Common Data Types
      1.4.1. Id
      1.4.2. Int and UnsignedInt
      1.4.3. PatchObject
      1.4.4. Resource
      1.4.5. UTCDateTime
   1.5. Common Properties
      1.5.1. contexts
      1.5.2. extra
      1.5.3. label
      1.5.4. pref
      1.5.5. phonetic
   1.6. Internationalization
      1.6.1. Free-form text
      1.6.2. URIs
   1.7. Validating JSContact
      1.7.1. Case-Sensitivity
      1.7.2. IANA-registered Properties
      1.7.3. Unknown Properties
      1.7.4. Enumerated Values
   1.8. Vendor-Specific Extensions
      1.8.1. Vendor-specific Properties
      1.8.2. Vendor-specific Values
   1.9. Versioning
      1.9.1. Version Format and Requirements
      1.9.2. Current Version
2. Card
   2.1. Metadata Properties
      2.1.1. @type
      2.1.2. version
      2.1.3. created
      2.1.4. kind
      2.1.5. language
      2.1.6. members
      2.1.7. prodId
      2.1.8. relatedTo
2.1.9. uid
2.1.10. updated

2.2. Name and Organization Properties

2.2.1. name
2.2.2. organizations
2.2.3. speakToAs
2.2.4. titles

2.3. Contact Properties

2.3.1. emails
2.3.2. onlineServices
2.3.3. phones
2.3.4. preferredLanguages

2.4. Calendaring and Scheduling Properties

2.4.1. calendars
2.4.2. schedulingAddresses

2.5. Address and Location Properties

2.5.1. addresses

2.6. Resource Properties

2.6.1. cryptoKeys
2.6.2. directories
2.6.3. links
2.6.4. media

2.7. Multilingual Properties

2.7.1. localizations

2.8. Additional Properties

2.8.1. anniversaries
2.8.2. keywords
2.8.3. notes
2.8.4. personalInfo

3. IANA Considerations

3.1. Media Type Registration

3.2. Creation of the "JSContact" Registry Group

3.3. Registry Policy and Change Procedures

3.3.1. Preliminary Community Review
3.3.2. Submit Request to IANA
3.3.3. Designated Expert Review
3.3.4. Change Procedures

3.4. Creation of the "JSContact Version" Registry

3.4.1. "JSContact Version" Registry Template
3.4.2. Initial Contents for the "JSContact Version" Registry

3.5. Creation of the "JSContact Properties" Registry

3.5.1. "JSContact Properties" Registry Template
3.5.2. Initial Contents for the "JSContact Properties" Registry

3.6. Creation of the "JSContact Types" Registry

3.6.1. "JSContact Types" Registry Template
3.6.2. Initial Contents for the "JSContact Types" Registry

3.7. Creation of the "JSContact Enum Values" Registry

3.7.1. "JSContact Enum Values" Registry Property Template
3.7.2. "JSContact Enum Values" Registry Value Template
1. Introduction

This document defines a data model for contact card data normally used in address book or directory applications and services. It aims to be an alternative to the vCard data format [RFC6350].

The key design considerations for this data model are as follows:

*The data model and set of attributes should mostly be compatible with the one defined for the vCard data format [RFC6350] and extensions ([RFC6473], [RFC6474], [RFC6715], [RFC6869], [RFC8605]). The specification should add new attributes or value types where appropriate. Not all existing vCard definitions need an equivalent in JSContact, especially if the vCard definition is considered to be obsolete or otherwise inappropriate. Conversion between the data formats need not fully preserve semantic meaning.

*The attributes of the card data represented must be described as simple key-value pairs, reducing complexity of their representation.

*The data model should avoid all ambiguities and make it difficult to make mistakes during implementation.

*Extensions, such as new properties and components, MUST NOT lead to requiring an update to this document.

The representation of this data model is defined in the I-JSON format [RFC7493], which is a strict subset of the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. Using JSON is mostly a pragmatic choice: its widespread use makes JSContact easier to adopt, and the availability of production-ready JSON implementations eliminates a whole category of parser-related interoperability issues.

1.1. Motivation and Relation to vCard, jCard and xCard

The vCard data format [RFC6350] is an interchange format for contacts data between address book service providers and vendors.
However, this format has gone through multiple specifications iterations with only a subset of its deprecated version 3 [RFC2426] being widely in use. Consequently, products and services internally use a richer contact data model than they expose when serializing that information to vCard. In addition, service providers often use a proprietary JSON representation of contact data in their APIs.

JSContact provides a standard JSON-based data model and representation of contact data as an alternative to proprietary formats.

While writing this document, several features missing in vCard were brought to the attention of the authors, such as social media contacts, gender pronouns and others. This highlights how vCard is not perceived as an evolving format and consequently hasn't been updated since close to ten years. JSContact addresses these unmet demands and defines new vCard properties and parameters to allow interchanging them in both formats.

The xCard [RFC6351] and jCard [RFC7095] specifications define alternative representations for vCard data, in XML and JSON format respectively. Both explicitly aim to not change the underlying data model. Accordingly, they are regarded as equal to vCard in the context of this document.

1.2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The ABNF definitions in this document use the notations of [RFC5234]. ABNF rules not defined in this document either are defined in [RFC5234] (such as the ABNF for CRLF, WSP, DQUOTE, VCHAR, ALPHA, and DIGIT) or [RFC6350].

1.3. Data Type Notations

This section introduces the notations and terminology used to define data types in JSContact.

The underlying format for JSContact is JSON and so also its data types build on JSON values. The terms "object" and "array" as well as the four primitive types ("strings", "numbers", "booleans", and "null") are to be interpreted as described in Section 1 of [RFC8259]. All JSContact data MUST be valid according to the constraints given in I-JSON [RFC7493]. Unless otherwise noted, all member names in JSON objects and all string values are case-
Sensitive. Within context of JSON objects, the term "key" is synonymous with "member name" as defined in Section 1 of [RFC8259].

### 1.3.1. Objects and Properties

JSContact defines data types for contact information such as addresses or names. This information typically consists of multiple related elements, for example a personal name and surname together form a name. These related elements are organized in JSContact objects. A JSContact object is a JSON object which:

1. Has a unique type name registered in the IANA JSContact Types Registry (Section 3.6).
2. Has one or more object members for which the name and allowed value types are specified. Such members are called "properties".
3. Has one property named @type with a string value that matches the type name of this JSContact object. In general, this property does not need to be set explicitly as outlined in Section 1.3.4.

The following sections specify how to define JSContact object types. Section 1.7 and Section 1.8 then define the exact requirements for property names.

The next paragraph illustrates how a JSContact object is defined:

A Foo object has the following properties:

- *qux: Number (mandatory). Defines the qux-ishness of this contact. The value MUST be an integer greater than 0 and less than 10.

Here, a JSContact object type named Foo is defined. In addition to its @type property it has a property named qux for which values MUST be valid according to the definition of the Number type. The property has one attribute, mandatory, which specifies that the property MUST be present for an instance of this JSContact object to be valid. Finally, a free-text description describes the semantics and further restrictions.

### 1.3.2. Type Signatures

Type signatures are given for all JSON values and JSContact definitions in this document. The following conventions are used:

- *String - The JSON string type.
*Number - The JSON number type.

*Boolean - The JSON boolean type.

*A[B] - A JSON object where the keys are all of the type A, and the values are all of the type B.

*A[] - A JSON array of values of type A.

*A|B - The value is either of type A or of type B.

** - The type is undefined (the value could be any type, although permitted values may be constrained by the context of this value).

Section 1.4 defines common data types, including signed or unsigned integers and dates.

1.3.3. Property Attributes

Object properties may also have a set of attributes defined along with the type signature. These have the following meanings:

*mandatory: The property **MUST** be set for an instance of this object to be valid.

*optional: The property can but not need be set for an instance of this object to be valid.

*default: This is followed by a JSON value. That value will be used for this property if it is omitted.

*defaultType: This is followed by the name of a JSContact object type. A property value of JSContact object type is expected to be of this named type, in case it omits the @type property.

1.3.4. The @type Property

This property is defined as:

*@type: String. Specifies the type of this object. This **MUST** match the type name of the JSContact object of which this JSON object is an instance of.

The purpose of this property is to help implementations identify which JSContact object type a given JSON object represents. Implementations **MUST** validate that JSON objects with this property conform to the specification of the JSContact object type of that name.
In many cases the @type property value is implied by where its object occurs in JSContact data. Assuming that both A and B are JSContact object types:

*An object that is set as the value for a property with type signature A MAY have the @type property set. If the @type property is not set then its value is implied to be A by the property definition.

*An object that is set as the value for a property with type signature A|B (defaultType: A) MAY have the @type property set if it is an instance of A. It MUST have the @type property set if it is an instance of B. If instead the defaultType attribute is not defined then the @type property MUST also be set for A.

*An object that is not the value of a property, such as the root of JSON data (directly or as member of an array), MUST have the @type property set.

1.4. Common Data Types

In addition to the standard JSON data types, a couple of additional data types are common to the definitions of JSContact objects and properties.

1.4.1. Id

Where Id is given as a data type, it means a String of at least 1 and a maximum of 255 octets in size, and it MUST only contain characters from the URL and Filename Safe base64url alphabet, as defined in Section 5 of [RFC4648], excluding the pad character (=). This means the allowed characters are the ASCII alphanumeric characters (A-Za-z0-9), hyphen (-), and underscore (_).

In many places in JSContact a JSON map is used where the map keys are of type Id and the map values are all the same type of object. This construction represents an unordered set of objects, with the added advantage that each entry has a name (the corresponding map key). This allows for more concise patching of objects, and, when applicable, for the objects in question to be referenced from other objects within the JSContact object. The map keys MUST be preserved across multiple versions of the JSContact object.

Unless otherwise specified for a particular property, there are no uniqueness constraints on an Id value (other than, of course, the requirement that you cannot have two values with the same key within a single JSON map). For example, two Card (Section 2) objects might use the same Ids in their respective photos properties. Or within the same Card the same Id could appear in the emails and phones
properties. These situations do not imply any semantic connections among the objects.

1.4.2. Int and UnsignedInt

Where Int is given as a data type, it means an integer in the range \(-2^{53}+1 \leq \text{value} \leq 2^{53}-1\), the safe range for integers stored in a floating-point double, represented as a JSON Number.

Where UnsignedInt is given as a data type, it means an integer in the range \(0 \leq \text{value} \leq 2^{53}-1\), represented as a JSON Number.

1.4.3. PatchObject

A PatchObject is of type String[*], and represents an unordered set of patches on a JSON object. Each key is a path represented in a subset of JSON pointer format [RFC6901]. The paths have an implicit leading /, so each key is prefixed with / before applying the JSON pointer evaluation algorithm.

A patch within a PatchObject is only valid if all the following conditions apply:

1. The pointer MAY reference inside an array but if the last reference token in the pointer is an array index, then the patch value MUST NOT be null. The pointer MUST NOT use "." as an array index in any of its reference tokens (i.e., you MUST NOT insert/delete from an array, but you MAY replace the contents of its existing members. To add or remove members, one needs to replace the complete array value).

2. All reference tokens prior to the last (i.e., the value after the final slash) MUST already exist as values in the object being patched. If the last reference token is an array index, then a member at this index MUST already exist in the referenced array.

3. There MUST NOT be two patches in the PatchObject where the pointer of one is the prefix of the pointer of the other, e.g., addresses/1/city and addresses.

4. The value for the patch MUST be valid for the property being set (of the correct type and obeying any other applicable restrictions), or if null the property MUST be optional.

The value associated with each pointer determines how to apply that patch:

*If null, remove the property from the patched object. If the key is not present in the parent, this is a no-op.
*If non-null, set the value given as the value for this property (this may be a replacement or addition to the object being patched).

A PatchObject does not define its own @type (Section 1.3.4) property. Instead, a @type property in a patch MUST be handled as any other patched property value.

Implementations MUST reject in its entirety a PatchObject if any of its patches are invalid. Implementations MUST NOT apply partial patches.

1.4.4. Resource

This data type defines a resource associated with the entity represented by this Card, identified by a URI [RFC3986]. Several property definitions later in this document refer to the Resource data type as the basis for their property-specific value types. The Resource data type defines the properties that are common to all of them. Property definitions making use of Resource MAY define additional properties for their value types.

The @type property value MUST NOT be Resource, instead it MUST be the name of a concrete resource type (see Section 2.6). A Resource object has the following properties.

* @type: String. Specifies the type of this resource object. The allowed value is defined in later sections of this document for each concrete resource type (Section 2.6).

* kind: String (optional). The kind of the resource. The allowed values are defined in the property definition that makes use of the Resource type. Some property definitions may change this property from being optional to mandatory.

* uri: String (mandatory). The resource value. This MUST be a URI as defined in Section 3 of [RFC3986].

* mediaType: String (optional). Used for URI resource values. Provides the media type [RFC2046] of the resource identified by the URI.

* contexts: String[Boolean] (optional). The contexts in which to use this resource. Also see Section 1.5.1.

* pref: UnsignedInt (optional). The preference of this resource in relation to other resources. Also see Section 1.5.4.

* label: String (optional). A custom label for the value, see Section 1.5.3.
1.4.5. UTCDateTime

This is a string in [RFC3339] date-time format, with the further restrictions that any letters MUST be in uppercase, and the time offset MUST be the character Z. Fractional second values MUST NOT be included unless non-zero and MUST NOT have trailing zeros, to ensure there is only a single representation for each date-time.

For example, 2010-10-10T10:10:10.003Z is conformant, but 2010-10-10T10:10:10.000Z is invalid and is correctly encoded as 2010-10-10T10:10Z.

1.5. Common Properties

Most of the properties in this document are specific to a single JSContact object type. Such properties are defined along with the respective object type. The properties in this section are common to multiple data types and are defined here to avoid repetition. Note that these properties MUST only be set for a JSContact object if they are explicitly mentioned to be allowed for this object type.

1.5.1. contexts

Type: String[Boolean]

This property associates contact information with one or more contexts in which it should be used. For example, someone might have distinct phone numbers for work and private contexts, and may set the desired context on the respective phone number in the phones (Section 2.3.3) property.

This section defines common contexts. Additional contexts may be defined in the properties or data types that make use of this property. The enumerated (Section 1.7.4) common context values are:

*private: the contact information may be used in a private context.

*work: the contact information may be used in a professional context.

1.5.2. extra

This is a reserved property name. Implementations MUST NOT set this property in a JSContact object. Any JSContact object including a property with this name MUST be considered invalid.

The purpose of this reserved property name is to provide implementors with a name which is certain to never occur as a property name in a JSContact object. Implementations might want to
map unknown or vendor-specific properties to a variable with this name, but this is implementation-specific.

1.5.3. label

Type: String

This property allows associating contact data with user-defined labels. Such labels may be set for phone numbers, email addresses and resources. Typically, these labels are displayed along with their associated contact data in graphical user interfaces. Such labels best be succinct to properly display on small graphical interfaces and screens.

1.5.4. pref

Type: UnsignedInt

This property allows defining a preference order for contact information. For example, a person may have two email addresses and prefer to be contacted with one of them.

Its value MUST be in the range 1 and 100. Lower values correspond to a higher level of preference, with 1 being most preferred. If no preference is set, then the contact information MUST be interpreted as being least preferred.

Note that the preference only is defined in relation to contact information of the same type. For example, the preference orders within emails and phone numbers are independent of each other.

1.5.5. phonetic

This property defines how to pronounce a value in the language indicated in the Card language (Section 2.1.5) property or the language tag of its localizations (Section 2.7.1). Exemplary uses are to define how to pronounce Japanese names, or for romanization of Mandarin or Cantonese name and address components. The properties are defined as follows:

* phonetic: String. Contains the phonetic representation of a value. Any script language subtag in the Card language (Section 2.1.5) property MUST be ignored for use with the phonetic property. If this property is set, then at least one of the phoneticScript or phoneticSystem properties that relate to this value MUST be set.

* phoneticScript: String. The script used in the value of the related phonetic property. This MUST be a valid script subtag as defined in Section 2.2.3 of [RFC5646].
The phonetic system used in the related value of the phonetic property. The enumerated values (Section 1.7.4) are:

- **ipa**: denotes the International Phonetic Alphabet [IPA].
- **jyut**: denotes the Cantonese romanization system "Jyutping".
- **piny**: denotes the Standard Mandarin romanization system "Hanyu Pinyin".

The relation between the phoneticSystem, phoneticScript and phonetic properties is type-specific. This specification defines this relation in the **Name** (Section 2.2.1) and **Address** (Section 2.5.1) object types, respectively.

The following example illustrates the phonetic property for a **name** (Section 2.2.1):

```json
"name": {
  "components": [
    { "kind": "given", "value": "John", "phonetic": "/ˈdʒɔːn/" },
    { "kind": "surname", "value": "Smith", "phonetic": "/smiθ/" }
  ],
  "phoneticSystem": "ipa"
}
```

Figure 1: Example of phonetic for the name "John Smith" as pronounced in the USA.

### 1.6. Internationalization

JSContact aims to be used for international contacts and addressbook data. Notably text values such as names and addresses are likely to cover a wide range of languages and cultures. This section describes internationalization for free-form text values, as well as for Uniform Resource Identifiers (URIs).

#### 1.6.1. Free-form text

Properties having free-form text values **MAY** contain any valid sequence of Unicode characters encoded as a JSON string. Such values can contain unidirectional left-to-right and right-to-left text, as well as bidirectional text using Unicode Directional Formatting.
Characters described in Section 2 of [UBidi]. Implementations setting bidirectional text **MUST** make sure that each property value complies with the requirements of the Unicode Bidirectional Algorithm. Implementations **MUST NOT** assume that text values of adjacent properties are processed or displayed as a combined string, for example the values of a given name component and a surname component may or may not to be rendered together.

### 1.6.2. URIs

Several properties require their string value to be a URI as defined in [RFC3986]. Implementations **MUST** make sure to use proper percent-encoding for URIs that can not be represented using unreserved URI characters. Section 3.1 of [RFC3987] defines how to convert Internationalized Resource Identifiers to URIs. JSContact makes no recommendation how to display URIs, but section "4.8.3 Internationalization and special characters" of the W3C URL Standard [W3C-URL] provides guidance for URIs found in context of a web browser.

### 1.7. Validating JSContact

This specification distinguishes between three kinds of properties regarding validation: IANA-registered properties and unknown properties are defined in this section, while vendor-specific properties are defined in Section 1.8.1. A JSContact object is invalid if any of its properties are invalid.

This document defines for each property if it is mandatory or optional. A mandatory property **MUST** be present for a JSContact object to be valid. An optional property does not need to be present. The values of both required and optional properties **MUST** adhere to the data type and definition of that property.

#### 1.7.1. Case-Sensitivity

All property names, object type names and enumerated values are case-sensitive, if not explicitly stated otherwise in their according definition. Implementations **MUST** handle a JSContact object as invalid if a type name, property name or enumerated value only differs in case from one defined for any JSContact version known to that implementation. This applies regardless of what JSContact version the Card object defines in its `version` property. Section 1.7.3 defines how to handle unknown properties.

#### 1.7.2. IANA-registered Properties

An IANA-registered property is any property that has been registered according to the IANA property registry rules as outlined in Section 3. All properties defined in this specification, including
their object value types and enumerated values, are registered at IANA.

Implementations **MUST** validate IANA-registered properties in JSContact data, unless they are unknown to the implementation (see Section 1.7.3). They **MUST** reject invalid IANA-registered properties. A property is invalid if its name matches the name of an IANA-registered property but the value violates its definition according to the JSContact specification version defined in the Card version property (Section 2.1.2).

IANA-registered property names **MUST NOT** contain US-ASCII control characters (U+0000 to U+001F, U+007F), the COLON (U+003A) or QUOTATION MARK (U+0022) characters. They **MUST** only contain US-ASCII alphanumeric characters that match the ALPHA and DIGIT rules defined in Appendix B.1 of [RFC5234]) or the COMMERCIAL AT (U+0040) character. IANA-registered property names **MUST** be notated in lower camel case.

### 1.7.3. Unknown Properties

Implementations may encounter JSContact data where a property name is unknown to that implementation, but the name adheres to the syntactic restrictions of IANA-registered property names. Implementations **MUST** make sure that such a name does not violate the case-sensitivity rules defined in Section 1.7.1. If the property name is valid, then implementations **MUST NOT** treat such properties as invalid. Instead, they **MUST** preserve them in the JSContact object.

Implementations that create or update JSContact data **MUST** only set IANA-registered properties or vendor-specific properties. Preserving properties that are unknown to the implementation, is to allow applications and services to interoperate without data loss, even if not all of them implement the same set of JSContact extensions.

### 1.7.4. Enumerated Values

Several properties in this document restrict their allowed values to be from a list of String values. These values are case-sensitive. If not noted otherwise for a specific property, the initial list of values for such properties is registered at IANA in the JSContact Enum Values Registry (Section 3.7). Implementations **MUST** only set IANA-registered or vendor-specific (Section 1.8.2) values for such properties.

### 1.8. Vendor-Specific Extensions

Vendors may extend properties and values for experimentation or to store contacts data that only is useful for a single service or
application. Such extensions are not meant for interoperation. If instead interoperation is desired, vendors are strongly encouraged
to define and register new properties, types and values at IANA.
Section 3 defines how to register new properties, types or values at
IANA. Section 1.7.2 defines the naming conventions for IANA-
registered elements.

1.8.1. Vendor-specific Properties

Vendor-specific property names **MUST** start with a vendor-specific
prefix followed by a name, as produced by the v-extension ABNF
below. The prefix and name together form the property name. The
vendor-specific prefix **MUST** be a domain name under control of the
service or application that sets the property, but it need not
resolve in the Domain Name System [RFC1034] and [RFC1035]. The
prefix ietf.org and its subdomain names are reserved for IETF
specifications. The name **MUST NOT** contain the TILDE (U+007E) and
SOLIDUS (U+002F) characters, as these require special-escaping when
encoding a JSON Pointer [RFC6901] for that property.

Vendor-specific properties **MAY** be set in any JSContact object.
Implementations **MUST** preserve vendor-specific properties in
JSContact data, irrespective if they know their use. They **MUST NOT**
reject the property value as invalid, unless they are in control of
the vendor-specific property as outlined in the above paragraph.

The ABNF rule v-extension formally defines valid vendor-specific
property names. Note that the vendor prefix allows for more values
than are allowed as Internationalized Domain Names (IDN) [RFC8499].
This is to allow JSContact implementations simply validate property
names without implementing the full set of rules that apply to
domain names.

v-extension = v-prefix ":" v-name
v-prefix = v-label *("." v-label)
v-label = alnum-int / alnum-int *(alnum-int / ":") alnum-int
alnum-int = ALPHA / DIGIT / NON-ASCII
    ; see RFC 6350 Section 3.3
v-name = 1*(WSP / "!" / %x23-2e / %x30-7d / NON-ASCII)
    ; any characters except CTLs, DQUOTE, SOLIDUS and TILDE

Figure 2: ABNF rules for vendor-specific property names

The value of vendor-specific properties can be any valid JSON value,
and naming restrictions do not apply to such values. Specifically,
if the property value is a JSON object then the keys of such objects need not be named as vendor-specific properties. The example in Figure 3 illustrates this:

```
"example.com:foo": "bar",
"example.com:foo2": {
    "bar": "baz"
}
```

Figure 3: Examples of vendor-specific properties

### 1.8.2. Vendor-specific Values

Some JSContact IANA-registered properties allow their values to be vendor-specific. One such example is the kind property Section 2.1.4, which enumerates its standard values but also allows for arbitrary vendor-specific values. Such vendor-specific values MUST be valid v-extension values as defined in Section 1.8.1. The example in Figure 4 illustrates this:

```
"kind": "example.com:baz"
```

Figure 4: Example of a vendor-specific value

Vendors are strongly encouraged to specify a new standard value once a vendor-specific one turns out to be useful also for other systems.

### 1.9. Versioning

Every instance of a JSContact Card (Section 2) indicates which JSContact version its IANA-registered properties and values are based on. The version is indicated both in the version (Section 2.1.2) property within the Card and in the version (Section 3.1) parameter of the JSContact MIME content type. All IANA-registered elements indicate the version at which they got introduced or obsoleted.

#### 1.9.1. Version Format and Requirements

A JSContact version consists of a numeric major and minor version, separated by the FULL STOP character (U+002E). Later versions are numerically higher than former versions, with the major version being more significant than the minor version. A version value is produced by the ABNF

```
jsversion = 1*DIGIT "." 1*DIGIT
```
Differing major version values indicate substantial differences in JSContact semantics and format. Implementations **MUST** be prepared that property definitions and other JSContact elements differ in a backwards-incompatible manner.

Differing minor version values indicate additions that enrich JSContact data, but do not introduce backwards-incompatible changes. Typically, these are new property enum values or properties with narrow semantic scope. A new minor version **MUST NOT** require implementations to change their processing of JSContact data. Changing the major version number resets the minor version number to zero.

### 1.9.2. Current Version

This specification registers JSContact version value 1.0 ([Table 1](#)).

### 2. Card

This section defines the JSContact object type Card. A Card stores contact information, typically that of a person, organization or company.

Its media type is defined in [Section 3.1](#).

*Figure 5* basic Card for the person "John Doe". As the object is the topmost object in the JSON data it has the @type property set according to the rules defined [Section 1.3.4](#).

```json
{
  "@type": "Card",
  "version": "1.0",
  "uid": "22B2C7DF-9120-4969-8460-05956FE6B065",
  "kind": "individual",
  "name": {
    "components": [
      { "kind": "given", "value": "John" },
      { "kind": "surname", "value": "Doe" }
    ],
    "isOrdered": true
  }
}
```

*Figure 5: Example of a basic Card*
2.1. **Metadata Properties**

This section defines properties about this instance of a Card, such as its unique identifier, its creation date, how it relates to other Cards and other metadata information.

2.1.1. **@type**

Type: String (mandatory).

This **MUST** be Card, if set.

2.1.2. **version**

Type: String (mandatory).

Specifies the JSContact version used to define this Card. The value **MUST** be one of the IANA-registered JSContact Enum Values for the version property. Also see Section 1.9.2.

```
"version": "1.0"
```

Figure 6: version example

2.1.3. **created**

Type: UTCDateTime (optional).

The date and time when this Card was created.

```
"created": "2022-09-30T14:35:10Z"
```

Figure 7: created example

2.1.4. **kind**

Type: String (optional, default: individual). The kind of the entity the Card represents.

The enumerated (Section 1.7.4) values are:

* individual: a single person
* group: a group person of persons or entities
* org: an organization
* location: a named location
*device: a device, such as appliances, computers, or network elements

*application: a software application

"kind": "individual"

Figure 8: kind example

2.1.5. language

Type: String (optional).

This is the language tag, as defined in [RFC5646], that best describes the language used for text in the Card, optionally including additional information such as the script. Note that values MAY be localized in the localizations property Section 2.7.1.

"language": "de-AT"

Figure 9: language example

2.1.6. members

Type: String[Boolean] (optional).

This identifies the set of Cards that are members of this group Card. Each key in the set is the uid property value of the member, each boolean value MUST be true. If this property is set, then the value of the kind property MUST be group.

The opposite is not true. A group Card will usually contain the members property to specify the members of the group, but it is not required to. A group Card without the members property can be considered an abstract grouping, or one whose members are known empirically (e.g., "IETF Participants").

"kind": "group",
"name": {
   "full": "The Doe family"
},
"uid": "urn:uuid:ab4310aa-fa43-11e9-8f0b-362b9e155667",
"members": {
   "urn:uuid:03a0e51f-d1aa-4385-8a53-e29025acd8af": true,
   "urn:uuid:b8767877-b4a1-4c70-9acc-505d3819e519": true
}
2.1.7. prodId

Type: String (optional).

The identifier for the product that created the Card. If set, the value MUST be at least one character long.

"prodId": "ACME Contacts App version 1.23.5"

2.1.8. relatedTo

Type: String[Relation] (optional).

Relates the object to other Cards. This is represented as a map, where each key is the uid of the related Card and the value defines the relation. The Relation object has the following properties:

* @type: String. This MUST be Relation, if set.

* relation: String[Boolean] (optional, default: empty Object) Describes how the linked object is related to the linking object. The relation is defined as a set of relation types. The key in the set defines the relation type, the value for each key in the set MUST be true. The relationship between the two objects is undefined if the set is empty.

The initial list of enumerated (Section 1.7.4) relation types matches the IANA-registered TYPE parameter [IANAvCard] values of the vCard RELATED property (Section 6.6.6 of [RFC6350]):

- acquaintance
- agent
- child
- colleague
- contact
- co-resident
- co-worker
- crush
- date
- emergency
- friend
- kin
- me
- met
- muse
- neighbor
- parent
- sibling
- spouse
- sweetheart

"relatedTo": {
  "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6": {
    "relation": {
      "friend": true
    }
  },
  "8cacdfb7d1ffdb59@example.com": {
    "relation": {}
  }
}

Figure 12: relatedTo example

2.1.9. uid

Type: String (mandatory).

An identifier, used to associate the object as the same across different systems, address books and views. The value SHOULD be a URN [RFC8141] but for compatibility with [RFC6350] it MAY also be a URI [RFC3986] or free-text value. The value of the URN SHOULD be in the uuid namespace [RFC4122]. As of this writing, a revision [I-D.ietf-uuidrev-rfc4122bis] of the UUID standard document is being worked on and is likely to introduce new UUID versions and best practices to generate global unique identifiers. Implementors SHOULD
follow any recommendations described there. Until then, implementations **SHOULD** generate identifiers using the random or pseudo-random UUID version described in Section 4.4 of [RFC4122].

"uid": "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"

Figure 13: uid example

### 2.1.10. updated

**Type:** UTCDateTime (optional).

The date and time when the data in this Card was last modified.

"updated": "2021-10-31T22:27:10Z"

Figure 14: updated example

### 2.2. Name and Organization Properties

This section defines properties that name the entity represented by this Card, its related organizations and roles, as well as how to refer the entity represented by this Card in spoken or written language.

#### 2.2.1. name

**Type:** Name (optional).

The name of the entity represented by this Card. This can be any type of name, e.g., it can but need not be the legal name of a person.

#### 2.2.1.1. Name object

A Name object has the following properties

*`@type`: String. This **MUST** be Name, if set.

*`components`: NameComponent[] (optional). The components (Section 2.2.1.2) making up this name. This property **MUST** be set if the full property is not set, otherwise it **SHOULD** be set. The component list **MUST** have at least one entry having a different kind than separator.

Name components **SHOULD** be ordered such that their values joined as a String produce a valid full name of this entity. If so, implementations **MUST** set the isOrdered property value to true.
If the name components are ordered, then the defaultSeparator property and name components of kind separator give guidance on what characters to insert between components, but implementations are free to choose any others. In lack of a separator, inserting a single Space character in between name component values is a good choice.

If instead the name components follow no particular order, then the isOrdered property value MUST be false, the components property MUST NOT contain a NameComponent of kind separator and the defaultSeparator property MUST NOT be set.

Figure 15 is an example for the name "Vincent van Gogh". Note how a single name component value may consist of multiple words. Figure 16 illustrates a name with a second surname, such as a Spanish name. Additional examples are shown in Figure 18 and Figure 38.

```
"name": {
    "components": [
        { "kind": "given", "value": "Vincent" },
        { "kind": "surname", "value": "van Gogh" }
    ],
    "isOrdered": true
}
```

Figure 15: Example for a surname with two words

```
"name": {
    "components": [
        { "kind": "given", "value": "Diego" },
        { "kind": "surname", "value": "Rivera" },
        { "kind": "surname2", "value": "Barrientos" }
    ],
    "isOrdered": true
}
```

Figure 16: Example for a second surname

*isOrdered: Boolean (optional, default: false). This indicates if the name component sequence in the components property is ordered.

*defaultSeparator: String (optional). The default separator to insert between name component values when concatenating all name component values to a single String. Also see the definition of the separator kind for the NameComponent object. This property MUST NOT be set if the Name isOrdered property value is false or if the components property is not set.
*full: String (optional). This is the full name representation of this Name. This property **MUST** be set if the components property is not set.

"full": "Mr. John Q. Public, Esq."

Figure 17: full example

*sortAs: String[String] (optional).

This defines how this name lexicographically sorts in relation to other names when compared by a name component type. The key in the map defines the name component type. The value for that key defines the verbatim string to compare when sorting by this name component type. Absence of a key indicates that this name component type **SHOULD NOT** be considered during sort. Sorting by that missing name component type or if the sortAs property is not set is implementation-specific. This property **MUST NOT** be set if the components property is not set.

Each key in the map **MUST** be a valid name component type value as defined for the kind property of the NameComponent object (see below). For each key in the map there **MUST** exist at least one NameComponent object having that type in the components property of this name.

**Figure 18** illustrates the use of sortAs. The property value indicates that the middle name followed by both surnames should be used when sorting this name by surname. The absence of the middle indicates that the middle name on its own should be disregarded during sort. Even though the name only contains one name component for the given name, the sortAs property still explicitly defines how to sort by given name as otherwise sorting by it would be undefined.

*phoneticScript: String (optional). The script used in the value of the NameComponent phonetic property. Also see [Section 1.5.5](#). See **Figure 19** for an example.

*phoneticSystem: String (optional). The phonetic system used in the NameComponent phonetic property. Also see [Section 1.5.5](#). See **Figure 19** for an example.
"name": {
    "components": [
        { "kind": "given", "value": "Robert" },
        { "kind": "given2", "value": "Pau" },
        { "kind": "surname", "value": "Shou Chang" }
    ],
    "sortAs": {
        "surname": "Pau Shou Chang",
        "given": "Robert"
    },
    "isOrdered": true
}

Figure 18: Example for sortAs

{
    "@type": "Card",
    "language": "zh-Hant",
    "name": {
        "components": [
            { "kind": "surname", "value": " " },
            { "kind": "given", "value": " " },
            { "kind": "given2", "value": " " },
            { "kind": "given2", "value": " " }
        ],
        "localizations": {
            "yue": {
                "name/phoneticSystem": "jyut",
                "name/phoneticScript": "Latn",
                "name/components/0/phonetic": "syun1",
                "name/components/1/phonetic": "zungisaan1",
                "name/components/2/phonetic": "man4",
                "name/components/3/phonetic": "jat6sin1"
            }
        }
    }
}

Figure 19: Example for phonetic and localizations

2.2.1.2. NameComponent

A NameComponent object has the following properties:

* @type: String. This **MUST** be NameComponent, if set.

* value: String (mandatory). The value of this name component. This can be composed of one or multiple words, such as "Poe" or "van Gogh".
kind: String (mandatory). The kind of this name component. The enumerated (Section 1.7.4) values are:

- title: an honorific title or prefix, e.g., "Mr", "Ms", "Dr".

- given: a given name, also known as "first name", "personal name".

- given2: a name that appears between the given and surname, such as a middle name or patronymic name.

- surname: a surname, also known as "last name", "family name".

- surname2: a secondary surname (used in some cultures), also known as "maternal surname".

- credential: a credential, also known as "accreditation qualifier" or "honorific suffix", e.g., "B.A.", "Esq.".

- generation: a generation marker or qualifier, e.g., “Jr.” or “III”.

- separator: a formatting separator between two ordered name non-separator components. The value property of the component includes the verbatim separator, for example a hyphen character or even an empty string. This value has higher precedence than the defaultSeparator property of the Name. Implementations MUST NOT insert two consecutive separator components, instead they SHOULD insert a single separator component with the combined value. This component kind MUST NOT be set if the Name isOrdered property value is false.

*phonic: String (optional). This defines how to pronounce this name component. If this property is set, then at least one of the Name object phoneticSystem or phoneticScript properties MUST be set. Also see Section 1.5.5.

2.2.1.3. nicknames

Type: Id[Nickname] (optional).

The nicknames of the entity represented by this Card. A Nickname object has the following properties:

* @type: String. This MUST be Nickname, if set.

* name: String (mandatory). The nickname.

* contexts: String[Boolean] (optional) The contexts in which to use this nickname. Also see Section 1.5.1.
*pref: UnsignedInt (optional). The preference of this nickname in relation to other nicknames. Also see Section 1.5.4.

```
"nicknames": {
  "k391": {
    "name": "Johnny"
  }
}
```

Figure 20: nicknames example

2.2.2. organizations

Type: Id[Organization] (optional).

The companies or organizations names and units associated with this Card. An Organization object has the following properties, of which at least one of the name and units properties MUST be set:

* @type: String. This MUST be Organization, if set.

* name: String (optional). The name of this organization.

* units: OrgUnit[] (optional). A list of organizational units, ordered descending by hierarchy (e.g., a geographic or functional division sorts before a department within that division). If set, the list MUST contain at least one entry.

* sortAs: String (optional). This defines how this organization name lexicographically sorts in relation to other organizations when compared by name. The value defines the verbatim string value to compare. In absence of this property, the name property value MAY be used for comparison.

* contexts: String[Boolean] (optional). The contexts in which association with this organization apply. For example, membership in a choir may only apply in a private context. Also see Section 1.5.1.

A OrgUnit object has the following properties:

* @type: String. This MUST be OrgUnit, if set.

* name: String (mandatory). The name of this organizational unit.

* sortAs: String (optional). This defines how this organization unit name lexicographically sorts in relation to other organizational units of the same level when compared by name. The level is defined by the array index of this organizational unit
in the units property of the Organization object. The property value defines the verbatim string value to compare. In absence of this property, the name property value **MAY** be used for comparison.

```
"organizations": {
  "o1": {
    "name": "ABC, Inc.",
    "units": [
      { "name": "North American Division" },
      { "name": "Marketing" }
    ],
    "sortAs": "ABC"
  }
}
```

Figure 21: organizations example

2.2.3. **speakToAs**

Type: SpeakToAs (optional).

Provides information how to address, speak to or refer to the entity that is represented by this Card. A SpeakToAs object has the following properties, of which at least one of the grammaticalGender and pronouns properties **MUST** be set:

* @type: String. This **MUST** be SpeakToAs, if set.

* grammaticalGender: String (optional). Defines which grammatical gender to use in salutations and other grammatical constructs. For example, the German language distinguishes by grammatical gender in salutations such as "Sehr geehrte" (feminine) and "Sehr geeehrter" (masculine). The enumerated (Section 1.7.4) values are:

  - animate
  - common
  - feminine
  - inanimate
  - masculine
  - neuter

Note that the grammatical gender does not allow inferring the gender identities or assigned sex of the contact.
*pronouns: Id[Pronouns] (optional). Defines the pronouns that the contact chooses to use for themselves.

A Pronouns object has the following properties:

*@type: String. This **MUST** be Pronouns, if set.

*pronouns: String (mandatory). Defines the pronouns. Any value or form is allowed. Examples in English include she/her and they/them/their. The value **MAY** be overridden in the localizations property (Section 2.7.1).

*contexts: String[Boolean] (optional). The contexts in which to use these pronouns. Also see Section 1.5.1.

*pref: UnsignedInt (optional). The preference of these pronouns in relation to other pronouns in the same context. Also see Section 1.5.4.

```
"speakToAs": {
  "grammaticalGender": "neuter",
  "pronouns": {
    "k19": {
      "pronouns": "they/them",
      "pref": 2
    },
    "k32": {
      "pronouns": "xe/xir",
      "pref": 1
    }
  }
}
```

Figure 22: speakToAs example

### 2.2.4. titles

**Type : Id[Title] (optional).**

The job titles or functional positions of the entity represented by this Card. A Title object has the following properties:

*@type: String. This **MUST** be Title, if set.

*name: String (mandatory). The title or role name of the entity represented by this Card.

*kind: String (optional, default title). Describes the organizational or situational kind of this title. Some
organizations and individuals distinguish between titles as organizational positions and roles as more temporary assignments, such as in project management.

The enumerated (Section 1.7.4) values are:

- title
- role

*organizationId: Id (optional). The identifier of the organization in which this title is held.

"titles": {
  "le9": {
    "kind": "title",
    "name": "Research Scientist"
  },
  "k2": {
    "kind": "role",
    "name": "Project Leader",
    "organizationId": "o2"
  }
},
"organizations": {
  "o2": {
    "name": "ABC, Inc."
  }
}

Figure 23: titles example

2.3. Contact Properties

This section defines properties how to contact the entity represented by this Card.

2.3.1. emails

Type: Id[EmailAddress] (optional).

The email addresses to contact the entity represented by this Card. An EmailAddress object has the following properties:

* @type: String. This MUST be EmailAddress, if set.

* address: String (mandatory). The email address. This MUST be an addr-spec value as defined in Section 3.4.1 of [RFC5322].
*contexts: String[Boolean] (optional). The contexts in which to use this email address. Also see Section 1.5.1.

*pref: UnsignedInt (optional). The preference of this email address in relation to other email addresses. Also see Section 1.5.4.

*label: String (optional). A custom label for the value, see Section 1.5.3.

"emails": {
    "e1": {
        "contexts": {
            "work": true
        },
        "address": "jqpublic@xyz.example.com"
    },
    "e2": {
        "address": "jane_doe@example.com",
        "pref": 1
    }
}

Figure 24: emails example

2.3.2. onlineServices

Type: Id[OnlineService] (optional).

The online services that are associated with the entity represented by this Card. This can be messaging services, social media profiles, and other. An OnlineService object has the following properties, of which at least the uri or user property MUST be set:

*@type: String. This MUST be OnlineService, if set.

*service: String (optional). The name of the online service or protocol. The name MAY be capitalized the same as on the service's website, app or publishing material, but names MUST be considered equal is they match case-insensitively. Examples are GitHub, kakao, Mastodon.

*uri: String (optional). This identifies the entity represented by this card at the online service. This MUST be a URI as defined in Section 3 of [RFC3986].

*user: String (optional). This names the entity represented by this Card at the online service. Any free-text value is allowed. The service property SHOULD be set.
*contexts: String[Boolean] (optional). The contexts in which to use this service. Also see Section 1.5.1.

*pref: UnsignedInt (optional). The preference of this service in relation to other services. Also see Section 1.5.4.

*label: String (optional). A custom label for the value, see Section 1.5.3.

"onlineServices": {
  "x1": {
    "uri": "xmpp:alice@example.com"
  },
  "x2": {
    "service": "Mastodon",
    "user": "@alice@example2.com",
    "uri": "https://example2.com/@alice"
  }
}

Figure 25: onlineServices example

2.3.3. phones

Type: Id[Phone] (optional).

The phone numbers to contact the entity represented by this Card. A Phone object has the following properties:

  *@type: String. This MUST be Phone, if set.

  *number: String (mandatory). The phone number, as either a URI or free-text. Typical URI schemes are the [RFC3966] tel or [RFC3261] sip schemes, but any URI scheme is allowed.

  *features: String[Boolean] (optional). The set of contact features that this phone number may be used for. The set is represented as an object, with each key being a method type. The boolean value MUST be true. The enumerated (Section 1.7.4) method type values are:

    - mobile: the number is for a mobile phone.

    - voice: the number is for calling by voice.

    - text: the number supports text messages (SMS).

    - video: the number supports video conferencing.
- **main-number**: this number is the main phone number, such as the number of the front-desk at a company, as opposed to a direct-dial number of an individual employee.

- **textphone**: the number is for a device for people with hearing or speech difficulties.

- **fax**: the number is for sending faxes.

- **pager**: the number is for a pager or beeper.

*contexts*: String[Boolean] (optional). The contexts in which to use this number. Also see Section 1.5.1.

*pref*: UnsignedInt (optional). The preference of this number in relation to other numbers. Also see Section 1.5.4.

*label*: String (optional). A custom label for the value, see Section 1.5.3.

```json
"phones": {
  "tel0": {
    "contexts": {
      "private": true
    },
    "features": {
      "voice": true
    },
    "number": "tel:+1-555-555-5555;ext=5555",
    "pref": 1
  },
  "tel3": {
    "contexts": {
      "work": true
    },
    "number": "tel:+1-201-555-0123"
  }
}
```

Figure 26: phones example

### 2.3.4. preferredLanguages

Type : Id[LanguagePref] (optional).

Defines the preferred languages for contacting the entity associated with this Card.
A LanguagePref object has the following properties:

* `@type: String. This MUST be LanguagePref, if set.`

* `language: String (mandatory). The preferred language. This MUST a language tag as defined in [RFC5646].`

* `contexts: String[Boolean] (optional). Defines the contexts in which to use this language. Also see Section 1.5.1.`

* `pref: UnsignedInt (optional). Defines the preference of this language in relation to other languages of the same contexts. Also see Section 1.5.4.`

```
"preferredLanguages": {
  "l1": {
    "language": "en",
    "contexts": {
      "work": true
    },
    "pref": 1
  },
  "l2": {
    "language": "fr",
    "contexts": {
      "work": true
    },
    "pref": 2
  },
  "l3": {
    "language": "fr",
    "contexts": {
      "private": true
    }
  }
}
```

Figure 27: preferredLanguages example

### 2.4. Calendaring and Scheduling Properties

This section defines properties how to schedule calendar events with the entity represented by this Card.

#### 2.4.1. calendars

Type: Id[Calendar] (optional).
These are resources for calendaring, such as calendars to lookup free-busy information for the entity represented by this Card. A Calendar object has all properties of the Resource (Section 1.4.4) data type, with the following additional definitions:

*The @type property value **MUST** be Calendar, if set.

*The kind property is mandatory. Its enumerated (Section 1.7.4) values are:

- calendar: the resource is a calendar that contains entries such as calendar events or tasks.
- freeBusy: the resource allows for free-busy lookups, for example to schedule group events.

"calendars": {
  "calA": {
    "kind": "calendar",
    "uri": "webcal://calendar.example.com/calA.ics"
  },
  "project-a": {
    "kind": "freeBusy",
    "uri": "https://calendar.example.com/busy/project-a"
  }
}

Figure 28: calendars example

2.4.2. schedulingAddresses

Type: Id[SchedulingAddress] (optional).

The scheduling addresses by which the entity may receive calendar scheduling invitations. A SchedulingAddress object has the following properties:

* @type: String. This **MUST** be SchedulingAddress, if set.

* uri: String (mandatory). The address to use for calendar scheduling with this contact. This **MUST** be a URI as defined in Section 3 of [RFC3986].

* contexts: String[Boolean] (optional). The contexts in which to use this scheduling address. Also see Section 1.5.1.

* pref: UnsignedInt (optional). The preference of this scheduling address in relation to other scheduling address. Also see Section 1.5.4.
*label: String (optional). A custom label for the scheduling address, see Section 1.5.3.

"schedulingAddresses": {
  "sched1": {
    "uri": "mailto:janedoe@example.com"
  }
}

Figure 29: schedulingAddresses example

2.5. Address and Location Properties

This section defines properties for postal addresses and geographical locations associated with the entity represented by this Card.

2.5.1. addresses

Type: Id[Address] (optional).

A map of address identifiers to Address objects, containing physical locations.

2.5.1.1. Address object

An Address object has the following properties:

* @type: String. This MUST be Address, if set.

* components: AddressComponent[] (optional). The components (Section 2.5.1.2) making up this address. This property MUST be set if the full property is not set, otherwise it SHOULD be set. The component list MUST have at least one entry having a different kind than separator.

Address components SHOULD be ordered such that their values joined as a String produce a valid full address. If so, implementations MUST set the isOrdered property value to true.

If the address components are ordered, then the defaultSeparator property and address components of kind separator give guidance what characters to insert between components, but implementations are free to choose any others. In lack of a separator, inserting a single Space character in between address component values is a good choice.

If instead the address components follow no particular order, then the isOrdered property value MUST be false, the components
property **MUST NOT** contain a AddressComponent of kind separator and the defaultSeparator property **MUST NOT** be set.

*isOrdered: Boolean (optional, default: false). This indicates if the address component sequence in the components property is ordered.

*countryCode: String (optional). The Alpha-2 country code [ISO.3166-1.2006].


*timeZone: String (optional). Identifies the time zone this address is located in. This **MUST** be a time zone name registered in the IANA Time Zone Database [IANATZ].

*contexts: String[Boolean] (optional). The contexts of the address information. The boolean value **MUST** be true. In addition to the common contexts (Section 1.5.1), allowed key values are:

- billing: an address to be used for billing.
- delivery: an address to be used for delivering physical items.

*full: String (optional). This is the full address, including street, region or country. The purpose of this property is to define an address, even if the individual address components are not known. If the street property is set, the full property **SHOULD NOT** be set.

*defaultSeparator: String (optional). The default separator to insert between address component values when concatenating all address component values to a single String. Also see the definition of the separator kind for the AddressComponent object. This property **MUST NOT** be set if the Address isOrdered property value is false or if the components property is not set.

*pref: UnsignedInt (optional). The preference of this address in relation to other addresses. Also see Section 1.5.4.

*phoneticScript: String (optional). The script used in the value of the AddressComponent phonetic property. Also see Section 1.5.5.

*phoneticSystem: String (optional). The phonetic system used in the AddressComponent phonetic property. Also see Section 1.5.5.

The following example illustrates the use of the address property. Additional examples are in Section 2.5.1.3.
2.5.1.2. AddressComponent object

An AddressComponent object has the following properties:

* @type: String. This MUST be AddressComponent, if set.

* value: String (mandatory). The value of this address component.

* kind: String (mandatory). The kind of this address component. The enumerated (Section 1.7.4) values are:

  - room: the room or suite number or identifier.

  - apartment: the extension designation, such as apartment number or unit or box number.

  - floor: the floor or level this address is located on.

  - building: the building, tower, or condominium this address is located in.

  - number: the street number, e.g., "123". This value is not restricted to numeric values, and can include any value such as number ranges ("112-10"), grid style ("39.2 RD"), alphanumerics ("N6W23001") or fractionals ("123 1/2").
- name: the street name.

- block: the block name or number.

- subdistrict: the subdistrict, ward or other subunit of a district.

- district: the district name.

- locality: the municipality, city, town, village, post-town, or another locality.

- region: the administrative area, such as province, state, prefecture, county, canton.

- postcode: the postal code, post code, ZIP code or other short code associated with the address by the relevant country's postal system.

- country: the country name.

- direction: the Cardinal direction or quadrant, e.g., "North".

- landmark: the publically known prominent feature that can substitute the street name and number, e.g., White House, Taj Mahal.

- postOfficeBox: the post office box number or identifier.

- separator: a formatting separator between two ordered address non-separator components. The value property of the component includes the verbatim separator, for example a hyphen character or even an empty string. This value has higher precedence than the defaultSeparator property of the Address. Implementations MUST NOT insert two consecutive separator components, instead they SHOULD insert a single separator component with the combined value. This component kind MUST NOT be set if the Address isOrdered property value is false.

  * phonetic: String (optional). This defines how to pronounce this name component. If this property is set, then at least one of the Address object phoneticSystem or phoneticScript properties MUST be set. Also see Section 1.5.5.

2.5.1.3. Address examples

The following are examples of addresses, in addition to Figure 30.
Figure 31: Example of the address "46, 1 Sukhumvit 51 Alley, Khlong Tan Nuea, Watthana, Bangkok 10110, Thailand"
"addresses": {  
"k26": {  
  "components": [  
    { "kind": "block", "value": "2-7" },  
    { "kind": "separator", "value": "-" },  
    { "kind": "number", "value": "2" },  
    { "kind": "separator", "value": " " },  
    { "kind": "district", "value": "Marunouchi" },  
    { "kind": "locality", "value": "Chiyoda-ku" },  
    { "kind": "region", "value": "Tokyo" },  
    { "kind": "separator", "value": " " },  
    { "kind": "postcode", "value": "100-8994" }  
  ],  
  "defaultSeparator": ", "  
  "full": "2-7-2 Marunouchi, Chiyoda-ku, Tokyo 100-8994",  
  "isOrdered": true  
},  
"localizations": {  
"jp": {  
  "addresses/k26": {  
    "components": [  
      { "kind": "region", "value": " " },  
      { "kind": "locality", "value": " " },  
      { "kind": "district", "value": " " },  
      { "kind": "block", "value": "2-7" },  
      { "kind": "separator", "value": "-" },  
      { "kind": "number", "value": "2" },  
      { "kind": "postcode", "value": "100-8994" }  
    ],  
    "defaultSeparator": "",  
    "full": "100-8994 2-7-2",  
    "isOrdered": true  
  }  
}  
}  
}

Figure 32: Example of an address in Tokyo and its localization [Section 2.7.1] in Japanese.

2.6. Resource Properties

This section defines properties for digital resources associated with the entity represented by this Card.

2.6.1. cryptoKeys

Type: Id[CryptoKey] (optional).
These are cryptographic resources such as public keys and certificates associated with the entity represented by this Card. A CryptoKey object has all properties of the Resource (Section 1.4.4) data type, with the following additional definitions:

*The @type property value MUST be CryptoKey, if set.*

The following example shows how refer to an external cryptographic resource.

```
"cryptoKeys": {
  "mykey1": {
    "uri": "https://www.example.com/keys/jdoe.cer"
  }
}
```

Figure 33: cryptoKeys example with external data

The following example shows how to embed key data in the CryptoKey. The key data is depicted in multiple lines only for demonstration purposes.

```
"cryptoKeys": {
  "mykey2": {
    "uri": "data:application/pgp-keys;base64,LS0tLS1CRUdJTiBSU0EgUFVCTElDIEtFWS0tLS0tCk1JSUJDZ0tDQVFFQSt4R1ovd2N6OXVnRnBQMDd0c3BvN1uxN2wwUZweHhVNHBUazNMaWZ6OVIzenNJc3UKRVJ3dGE3K2ZXSWZ4T28yMDhlHQv1Wb2RTRXQzUUJHahDLYQm1weVdvEt3WjkzSEhhRFZaQUFMaS8yQ0oreFRCdF3WEdVwplRHZDmi9hWktia2ZqcE9pVUK4QwhMQwZqbwjRc9W0jFmUGgwWbUhTkNtcEN3Cm13UihB0VZobWh6K1BpQitEbWw0ViduS1cvVkhvMnVqVb4cTcr.TRIMmZueTNTZTNLWU9zRLGR1oxVE4KUVDZbEZ1U2hXckhQdG1MbVVkBX9QNN1NTDF0aytsNORJSXYClFoTFVLREFDZU01cm9NeDBrTGrhVVOlI4UAorMHVqMUMONEpSwmxDN3hGznFpTWJGUL5WjR0NL3SURBUUFCci0tlS0tRU5EIFJTQSBUMgS0VZLS0tLS0K"
  }
}
```

Figure 34: cryptoKeys example with embedded data

2.6.2. directories

Type: Id[Directory] (optional).

These are directory service resources, such as entries in a directory or organizational directories for lookup. A Directory
The Directory object has all properties of the Resource (Section 1.4.4) data type, with the following additional definitions:

*The @type property value **MUST** be Directory, if set.

*The kind property is mandatory. Its enumerated (Section 1.7.4) values are:

- **directory**: the resource is a directory service where the entity represented by this Card is part of. This typically is an organizational directory that also contains associated entities, e.g., co-workers and management in a company directory.

- **entry**: the resource is a directory entry of the entity represented by this Card. In contrast to the directory type, this is the specific URI for the entity within a directory.

In addition, the Directory object has the following property:

*listAs: UnsignedInt (optional). This defines the position of this directory resource in the list of all Directory objects having the same kind in this Card. If set, the listAs value **MUST** be higher than zero. Multiple directory resources **MAY** have the same listAs property value, or none. Sorting such entries is implementation-specific.

```
"directories": {
  "dir1": {
    "kind": "entry",
    "uri": "https://dir.example.com/addrbook/jdoe/Jean%20Dupont.vcf"
  },
  "dir2": {
    "kind": "directory",
    "uri": "ldap://ldap.example/o=Example%20Tech,ou=Engineering",
    "pref": 1
  }
}
```

Figure 35: directories example

2.6.3. **links**

Type: Id[Link] (optional).

These are links to resources that do not fit any of the other use-case specific resource properties. A Link object has all properties
of the Resource (Section 1.4.4) data type, with the following additional definitions:

*The @type property value **MUST** be Link, if set.

*The kind property is optional. Its enumerated (Section 1.7.4) values are:

- **contact**: the resource is a URI by which the entity represented by this Card may be contacted, including web forms or other media that require user interaction.

Figure 36: links example

```
"links": {
  "link3": {
    "kind": "contact",
    "uri": "mailto:contact@example.com",
    "pref": 1
  }
}
```

2.6.4. media

Type: Id[Media] (optional).

These are media resources such as photographs, avatars, or sounds associated with the entity represented by this Card. A Media object has all properties of the Resource (Section 1.4.4) data type, with the following additional definitions:

*The @type property value **MUST** be Media, if set.

*The kind property is mandatory. Its enumerated (Section 1.7.4) values are:

- **photo**: the resource is a photograph or avatar.

- **sound**: the resource is audio media, e.g., to specify the proper pronunciation of the name property contents.

- **logo**: the resource is a graphic image or logo associated with the entity represented by this Card.
2.7. Multilingual Properties

This section defines properties how to localize the content of this Card in human languages.

2.7.1. localizations

Type: String[PatchObject] (optional).

This localizes property values in this Card to languages other than the main language. Localizations provide language-specific alternatives for existing property values and **SHOULD NOT** add new properties.

The keys in the localizations property object are language tags [RFC5646]. The values are patch objects which localize the Card in the respective language tag. The paths in the PatchObject are relative to the Card that includes the localizations property. A patch **MUST NOT** target the localizations property.

Conceptually, a Card is localized as follows:

* Determine the language tag in which this Card should be localized in.

* If the localizations property includes a key for that language, obtain the PatchObject value. If there is no such key, stop.

* Create a copy of the Card, but do not copy the localizations property.

* Apply all patches in the PatchObject to the copy of the Card.

Figure 37: media example

```json
"media": {
  "res45": {
    "kind": "sound",
    "uri": "CID:JOHNQ.part8.19960229T080000.xyzMail@example.com"
  },
  "res47": {
    "kind": "logo",
    "uri": "https://www.example.com/pub/logos/abccorp.jpg"
  },
  "res1": {
    "kind": "photo",
    "uri": "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAASABIAAD/4...
  }
}
```
*Optionally, set the language property in the copy of the Card.

*Use the patched copy of the Card as the localized variant of the original Card.

A patch in the PatchObject may contain any value type. Its value **MUST** be a valid value according to the definition of the patched property.

**Figure 38** localizes the name property by completely replacing its contents in Ukrainian language with Cyrillic script.

```
{
  "name": {
    "components": [
      { "kind": "title", "value": "Mr." },
      { "kind": "given", "value": "Ivan" },
      { "kind": "given2", "value": "Petrovich" },
      { "kind": "surname", "value": "Vasiliev" }
    ]
  },
  "localizations": {
    "uk-Cyrl": {
      "name": {
        "components": [
          { "kind": "title", "value": "г-н" },
          { "kind": "given", "value": "Иван" },
          { "kind": "given2", "value": "Петрович" },
          { "kind": "surname", "value": "Васильев" }
        ]
      }
    }
  }
}
```

**Figure 38**: Example for localizing a top-level property

**Figure 39** localizes the title name by patching *inside* the titles property. All properties but the name property in the Title object are left as-is.
This section defines properties for which none of the previous sections are appropriate.

2.8.1. anniversaries

Type: Id[Anniversary] (optional).

These are memorable dates and events for the entity represented by this Card. An Anniversary object has the following properties:

* @type: String. This **MUST** be Anniversary, if set.

* kind: String (mandatory). Specifies the kind of the anniversary. The enumerated (Section 1.7.4) values are:

  - birth: a birthday anniversary
  - death: a deathday anniversary
  - wedding: a wedding day anniversary

* date: PartialDate|Timestamp (mandatory, defaultType: PartialDate).

  The date of this anniversary in the Gregorian calendar. This **MUST** either be a whole or partial calendar date or a complete UTC timestamp (see the definition of the Timestamp and PartialDate object types below).

* place: Address (optional). An address associated with this anniversary, e.g., the place of birth or death.
A PartialDate object represents a complete or partial calendar date in the Gregorian calendar. It represents either a complete date, or a year, or a month in a year, or a day in a month. It has the following properties, of which at least year or month and day MUST be set:

* @type: String. This MUST be PartialDate, if set.

* year: UnsignedInt (optional). This is the calendar year.

* month: UnsignedInt (optional). This is the calendar month, represented as the integers 1 <= month <= 12. If this property is set then either year or day MUST be set.

* day:UnsignedInt (optional). This is the calendar month day, represented as the integers 1 <= day <= 31, depending on the validity within the month and year. If this property is set then month MUST be set.

* calendarScale: String (optional). This is the calendar system in which this date occurs, in lowercase. This MUST be either a CLDR-registered calendar system name [RFC7529] or a vendor-specific value. The year, month and day still MUST be represented in the Gregorian calendar. Note that the year property might be required to convert the date between the Gregorian calendar and the respective calendar system.

A Timestamp object has the following properties:

* @type: String. This MUST be Timestamp, if set.

* utc: UTCDateTime (mandatory). Specifies the point in time in UTC time.

Figure 40 illustrates anniversaries with partial dates and timestamp. Note how the @type property is set for the Timestamp object value according to the rules defined Section 1.3.4.
"anniversaries": {  
  "k8": {  
    "kind": "birth",  
    "date": {  
      "year": 1953,  
      "month": 4,  
      "day": 15  
    }  
  },  
  "k9": {  
    "kind": "death",  
    "date": {  
      "@type": "Timestamp",  
      "utc": "2019-10-15T23:10:00Z"  
    },  
    "place": {  
      "full": "4445 Tree Street\nNew England, ND 58647\nUSA"  
    }  
  }  
}  

Figure 40: anniversaries example

2.8.2. keywords

Type: String[Boolean] (optional). A set of free-text keywords, also known as tags. The set is represented as an object, with each key being a keyword. The boolean value MUST be true.

"keywords": {  
  "internet": true,  
  "IETF": true  
}  

Figure 41: keywords example

2.8.3. notes

Type: Id[Note] (optional).

Free-text notes associated with this Card. A Note object has the following properties:

* @type: String. This MUST be Note, if set.
* note: String (mandatory). The free text value of this note.
* created: UTCDateTime (optional). The date and time when this note was created.
*author: Author (optional). The author of this note.

An Author object has the following properties, of which at least one other than @type MUST be set:

*@type: String. This MUST be Author, if set.

*name: String (optional). The name of this author.

*uri: String (optional). A URI value that identifies the author.

"notes": {
    "n1": {
        "note": "Open office hours are 1600 to 1715 EST, Mon-Fri",
        "created": "2022-11-23T15:01:32Z",
        "author": {
            "name": "John"
        }
    }
}

Figure 42: notes example

2.8.4. personalInfo

Type: Id[PersonalInfo] (optional).

Defines personal information about the entity represented by this Card. A PersonalInfo object has the following properties:

*@type: String. This MUST be PersonalInfo, if set.

*kind: String (mandatory). Specifies the kind of this personal information. The enumerated (Section 1.7.4) values are:

- expertise: a field of expertise or credential
- hobby: a hobby
- interest: an interest

*value: String (mandatory). The actual information.

*level: String (optional). Indicates the level of expertise, or engagement in hobby or interest. The enumerated (Section 1.7.4) values are:

- high
Type name: 

Subtype name: 

Required parameters:

- medium
- low

*listAs: UnsignedInt (optional). This defines the position of this personal information in the list of all PersonalInfo objects having the same kind in this Card. If set, the listAs value **MUST** be higher than zero. Multiple personal information entries **MAY** have the same listAs property value, or none. Sorting such entries is implementation-specific.

*label: String (optional). A custom label. See [Section 1.5.3](#).

```
"personalInfo": {
  "pi2": {
    "kind": "expertise",
    "value": "chemistry",
    "level": "high"
  },
  "pi1": {
    "kind": "hobby",
    "value": "reading",
    "level": "high"
  },
  "pi6": {
    "kind": "interest",
    "value": "r&b music",
    "level": "medium"
  }
}
```

Figure 43: personalInfo example

3. IANA Considerations

3.1. Media Type Registration

[[I-D.ietf-calext-jscontact](#)] defines a media type for use with JSContact data formatted in JSON.

Type name:

application

Subtype name:

jscontact+json

Required parameters:

None
Optional parameters:

version This parameter conveys the version of the JSContact data in the body part. It MUST NOT occur more than once. If this parameter is set, then the values of all JSContact version (Table 2) properties in the body part MUST match the parameter value.

Encoding considerations:
This is the same as the encoding considerations of application/json, as specified in Section 11 of [RFC8259].

Security considerations:
See Section 4 of [I-D.ietf-calext-jscontact].

Interoperability considerations:
While JSContact is designed to avoid ambiguities as much as possible, when converting objects from other contact formats to/from JSContact, it is possible that differing representations for the same logical data or ambiguities in interpretation might arise. The semantic equivalence of two JSContact objects may be determined differently by different applications, for example, where URL values differ in case between the two objects.

Published specification:
TBD

Applications that use this media type:
Applications that currently make use of the text/vcard media type can use this as an alternative.

Fragment identifier considerations:
A JSON Pointer fragment identifier may be used, as defined in [RFC6901], Section 6.

Additional information:
- Magic number(s): N/A
- File extensions(s): N/A
- Macintosh file type code(s): N/A

Person & email address to contact for further information:
calsify@ietf.org

Intended usage:
COMMON

Restrictions on usage:
N/A
3.2. Creation of the "JSContact" Registry Group

IANA is asked to create the "JSContact" registry group. The new registry definitions in the following sections all belong to that group.

3.3. Registry Policy and Change Procedures

Registry assignments that introduce backwards-incompatible (Section 1.9) changes require the JSContact major version to change, other changes only require to change the minor version. The registry policy for assignments that require the JSContact major version to change is Standards Action ([RFC8126], Section 4.9). The registry policy for other assignments is Specification Required ([RFC8126], Section 4.6).

The Designated Expert decides if a major or minor version change is required and assigns the new version to the Version Registry (Section 3.4). Version numbers increment by one, and a major version change resets the minor version to zero. An assignment may apply multiple changes and to more than one registry at once, in which case a single version change is sufficient. If the registry policy is Specification Required, then the Designated Expert may decide that it is enough to document the new assignment in the Description item of the respective registry.

A registration MUST have an intended usage of common, reserved, or obsolete.

*A common usage denotes an item with shared semantics and syntax across systems. Up-to-date systems MUST expect such items to occur in JSContact data.

*A reserved usage reserves an item in the registry without assigning semantics to avoid name collisions with future extensions or protocol use.

*An obsolete usage denotes an item that is no longer expected to be added by up-to-date systems. A new assignment has probably been defined covering the obsolete item's semantics.

The registration procedure is not a formal standards process but rather an administrative procedure intended to allow community
comment and check whether it is coherent without excessive time delay. It is designed to encourage vendors to document and register new items they add for use cases not covered by the original specification, leading to increased interoperability.

3.3.1. Preliminary Community Review

Notice of a potential new registration MUST be sent to the Calext mailing list <calsify@ietf.org> for review. This mailing list is appropriate to solicit community feedback on a proposed registry assignment.

The intent of the public posting to this list is to solicit comments and feedback on the choice of the item name or value, the unambiguity of its description, and a review of any interoperability or security considerations. The submitter may submit a revised registration proposal or abandon the registration completely at any time.

3.3.2. Submit Request to IANA

Registration requests can be sent to <iana@iana.org>.

3.3.3. Designated Expert Review

The primary concern of the designated expert (DE) is preventing name collisions and encouraging the submitter to document security and privacy considerations.

A new type name, property name or enumerated value MUST NOT differ only in case from an already registered name or value.

For a common-use registration, the DE is expected to confirm that suitable documentation is available to ensure interoperability. The DE should also verify that the new assignment does not conflict with work that is active or already published within the IETF.

The DE will either approve or deny the registration request and publish a notice of the decision to the Calext WG mailing list or its successor, as well as inform IANA. A denial notice must be justified by an explanation, and, in the cases where it is possible, concrete suggestions on how the request can be modified to become acceptable should be provided.

3.3.4. Change Procedures

Once a JSContact registry group item has been published by IANA, the change controller may request a change to its definition. The same procedure that would be appropriate for the original registration request is used to process a change request.
JSContact registrations do not get deleted; instead, items that are no longer believed appropriate for use are declared obsolete by a change to their "intended usage" field; such items will be clearly marked in the IANA registry.

Significant changes to a JSContact registry item's definition should be requested only when there are serious omissions or errors in the published specification, as such changes may cause interoperability issues. When review is required, a change request may be denied if it renders entities that were valid under the previous definition invalid under the new definition.

3.4. Creation of the "JSContact Version" Registry

IANA is asked to create the "JSContact Version" registry. The purpose of this new registry is to define the allowed value range of JSContact major and minor version numbers.

The registry entries sort numerically ascending by the "Major Version" column.

The registry process is outlined in Section 3.3.

3.4.1. "JSContact Version" Registry Template

**Major Version:** This is the numeric value of a JSContact major version number. It MUST be a positive integer.

**Highest Minor Version:** This is the maximum numeric value of a JSContact minor version for the given major version. It MUST be zero or a positive integer. All numbers less than or equal this value are valid minor version values for the given major version.

3.4.2. Initial Contents for the "JSContact Version" Registry

The following table lists the initial valid major and minor version number ranges.

<table>
<thead>
<tr>
<th>Major Version</th>
<th>Highest Minor Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: JSContact Versions

3.5. Creation of the "JSContact Properties" Registry

IANA is asked to create the "JSContact Properties" registry. The purpose of this new registry is to allow interoperability of extensions to JSContact objects.
The registry entries sort alphabetically ascending by the "Property Name" column first, "Property Context" second, "Since Version" third. Equal entries sort in any order.

The registry process for a new property is outlined in Section 3.3.

3.5.1. "JSContact Properties" Registry Template

Property Name: This is the name of the property. The property name **MUST NOT** already be registered for any of the object types listed in the "Property Context" field of this registration. Other object types **MAY** already have registered a different property with the same name; however, the same name **MUST** only be used when the semantics are analogous.

Property Type: This is the type of this property, using type signatures, as specified in Section 1.3.2. The property type **MUST** be registered in the "JSContact Types" registry.

Property Context: This is a comma-separated list of JSContact object types (Section 3.6.2) that contain this property.

Reference or Description: This is a brief description or RFC number and section reference where the property is specified (omitted for "reserved" property names). This must include references to all RFC documents where this property is introduced or updated.

Intended Usage: This may be "common", "reserved", or "obsolete".

Since Version: This defines the JSContact version on which this property definition is based on. The version **MUST** be one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

Until Version: This defines the JSContact version after which this property got obsoleted and **MUST NOT** be used in later versions. The Until Version value either **MUST NOT** be set, or be one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

Change Controller: This is who may request a change to this entry's definition (IETF for RFCs from the IETF stream).

3.5.2. Initial Contents for the "JSContact Properties" Registry

The following table lists the initial common usage entries of the "JSContact Properties" registry. The Since Version for all properties is 1.0. The Until Version for all properties is not set. All RFC section references are for [I-D.ietf-calext-jscontact]. The change controller for all these properties is IETF.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Property Context</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@type</td>
<td>String</td>
<td>NameComponent, Nickname, Note, OnlineService, Organization, OrgUnit, PartialDate, PersonalInfo, Phone, Pronouns, Relation, SchedulingAddress, SpeakToAs, Timestamp, Title</td>
<td>Section 2.5.1, Section 2.8.3, Section 2.2.3</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>Card</td>
<td>Section 2.1.2</td>
</tr>
<tr>
<td>address</td>
<td>String</td>
<td>EmailAddress</td>
<td>Section 2.3.1</td>
</tr>
<tr>
<td>addresses</td>
<td>Id[Address]</td>
<td>Card</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>anniversaries</td>
<td>Id[Anniversary]</td>
<td>Card</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>author</td>
<td>Author</td>
<td>Note</td>
<td>Section 2.8.3</td>
</tr>
<tr>
<td>calendars</td>
<td>Id[Calendar]</td>
<td>Card</td>
<td>Section 2.4.1</td>
</tr>
<tr>
<td>calendarScale</td>
<td>String</td>
<td>PartialDate</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>components</td>
<td>AddressComponent[]</td>
<td>Address</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>components</td>
<td>NameComponent[]</td>
<td>Name</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td>contexts</td>
<td>String[Boolean]</td>
<td>Media, Nickname, OnlineService, Organization, Phone, Pronouns, SchedulingAddress</td>
<td>Section 1.5.1, Section 2.8.4, Section 2.2.1, Section 2.4.1</td>
</tr>
<tr>
<td>coordinates</td>
<td>String</td>
<td>Address</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>countryCode</td>
<td>String</td>
<td>Address</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>created</td>
<td>UTCDateTime</td>
<td>Card, Note</td>
<td>Section 2.1.3, Section 2.8.1</td>
</tr>
<tr>
<td>date</td>
<td>PartialDate</td>
<td>Anniversary</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>day</td>
<td>UnsignedInt</td>
<td>PartialDate</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>defaultSeparator</td>
<td>String</td>
<td>Address, Name</td>
<td>Section 2.5.1, Section 2.6.2</td>
</tr>
<tr>
<td>directories</td>
<td>Id[Directory]</td>
<td>Card</td>
<td>Section 2.6.2</td>
</tr>
<tr>
<td>emails</td>
<td>Id[EmailAddress]</td>
<td>Card</td>
<td>Section 2.3.1</td>
</tr>
<tr>
<td>features</td>
<td>String[Boolean]</td>
<td>Phone</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>full</td>
<td>String</td>
<td>Address, Name</td>
<td>Section 2.5.1, Section 2.2.3</td>
</tr>
<tr>
<td>grammaticalGender</td>
<td>String</td>
<td>SpeakToAs</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>isOrdered</td>
<td>Boolean</td>
<td>Address, Name</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>Property Name</td>
<td>Property Type</td>
<td>Property Context</td>
<td>Reference or Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>keywords</td>
<td>String[Boolean]</td>
<td>Card, AddressComponent, Anniversary, Calendar, Card, CryptoKey, Directory, Link, Media, NameComponent, PersonalInfo, Title Calendar, CryptoKey, Directory, EmailAddress, Link, Media, OnlineService, PersonalInfo, Phone, SchedulingAddress</td>
<td>Section 2.8.2</td>
</tr>
<tr>
<td>kind</td>
<td>String</td>
<td>AddressComponent, Anniversary, Calendar, Card, CryptoKey, Directory, Link, Media, NameComponent, PersonalInfo, Title Calendar, CryptoKey, Directory, EmailAddress, Link, Media, OnlineService, PersonalInfo, Phone, SchedulingAddress</td>
<td>Section 2.5.1, Section 2.6.1</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Calendar, CryptoKey, Directory, EmailAddress, Link, Media, OnlineService, PersonalInfo, Phone, SchedulingAddress</td>
<td>Section 1.5.3, Section 2.8.4</td>
</tr>
<tr>
<td>language</td>
<td>String</td>
<td>Card, LanguagePref</td>
<td>Section 2.1.5, Section 2.8.4</td>
</tr>
<tr>
<td>level</td>
<td>String</td>
<td>PersonalInfo</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td>links</td>
<td>Id[Link]</td>
<td>Card</td>
<td>Section 2.6.3</td>
</tr>
<tr>
<td>listAs</td>
<td>UnsignedInt</td>
<td>Directory, PersonalInfo</td>
<td>Section 2.6.2, Section 2.8.4</td>
</tr>
<tr>
<td>localizations</td>
<td>String[PatchObject]</td>
<td>Card</td>
<td>Section 2.7.1</td>
</tr>
<tr>
<td>media</td>
<td>Id[Media]</td>
<td>Calendar, CryptoKey, Directory, Link, Media, OnlineService, PersonalInfo, Phone, SchedulingAddress</td>
<td>Section 2.6.4</td>
</tr>
<tr>
<td>mediaType</td>
<td>String</td>
<td>Calendar, CryptoKey, Directory, Link, Media</td>
<td>Section 1.4.4, Section 2.8.4</td>
</tr>
<tr>
<td>members</td>
<td>String[Boolean]</td>
<td>Card</td>
<td>Section 2.1.6</td>
</tr>
<tr>
<td>month</td>
<td>UnsignedInt</td>
<td>PartialDate</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>AddressComponent, Author, Nickname, Organization, OrgUnit, Title</td>
<td>Section 2.8.3, Section 2.8.4</td>
</tr>
<tr>
<td>nicknames</td>
<td>Id[Nickname]</td>
<td>Card</td>
<td>Section 2.2.1.3</td>
</tr>
<tr>
<td>note</td>
<td>String</td>
<td>Note</td>
<td>Section 2.8.3</td>
</tr>
<tr>
<td>notes</td>
<td>Id[Note]</td>
<td>Card</td>
<td>Section 2.8.3</td>
</tr>
<tr>
<td>number</td>
<td>String</td>
<td>Phone</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>onlineServices</td>
<td>Id[OnlineService]</td>
<td>Card</td>
<td>Section 2.3.2</td>
</tr>
<tr>
<td>organizationId</td>
<td>String</td>
<td>Title</td>
<td>Section 2.2.4</td>
</tr>
<tr>
<td>organizations</td>
<td>Id[Organization]</td>
<td>Card</td>
<td>Section 2.2.2</td>
</tr>
<tr>
<td>personalInfo</td>
<td>Id[PersonalInfo]</td>
<td>Card</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td>phones</td>
<td>Id[Phone]</td>
<td>Card</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>phonetic</td>
<td>String</td>
<td>AddressComponent, NameComponent</td>
<td>Section 2.5.1.2, Section 2.8.1</td>
</tr>
<tr>
<td>phoneticScript</td>
<td>String</td>
<td>Address, Name</td>
<td>Section 2.2.1, Section 2.8.1</td>
</tr>
<tr>
<td>phoneticSystem</td>
<td>String</td>
<td>Address, Name</td>
<td>Section 2.2.1, Section 2.8.1</td>
</tr>
<tr>
<td>place</td>
<td>Address</td>
<td>Address, Calendar, Address, Calendar, CryptoKey, Directory, EmailAddress, LanguagePref, Link, Media, Nickname, OnlineService, Phone,</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>pref</td>
<td>UnsignedInt</td>
<td>LanguagePref, Link, Media, Nickname, OnlineService, Phone, SchedulingAddress</td>
<td>Section 1.5.4, Section 2.8.4</td>
</tr>
</tbody>
</table>
### Table 2: JSContact Properties with "common" usage

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Property Context</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferredLanguages</td>
<td>String[LanguagePref]</td>
<td>Card</td>
<td>Section 2.3.4</td>
</tr>
<tr>
<td>prodId</td>
<td>String</td>
<td>Card</td>
<td>Section 2.1.7</td>
</tr>
<tr>
<td>pronouns</td>
<td>Id[Pronouns]</td>
<td>SpeakToAs</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>relatedTo</td>
<td>String[Relation]</td>
<td>Card</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>relation</td>
<td>String[Boolean]</td>
<td>Relation</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>schedulingAddresses</td>
<td>Id[SchedulingAddress]</td>
<td>Card</td>
<td>Section 2.4.2</td>
</tr>
<tr>
<td>service</td>
<td>String</td>
<td>OnlineService</td>
<td>Section 2.3.2</td>
</tr>
<tr>
<td>sortAs</td>
<td>String[String]</td>
<td>Name</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td>sortAs</td>
<td>String</td>
<td>Organization, OrgUnit</td>
<td>Section 2.2.2</td>
</tr>
<tr>
<td>speakToAs</td>
<td>SpeakToAs</td>
<td>Card</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>timeZone</td>
<td>String</td>
<td>Address</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>titles</td>
<td>Id[Title]</td>
<td>Card</td>
<td>Section 2.1.9</td>
</tr>
<tr>
<td>uid</td>
<td>String</td>
<td>Card</td>
<td>Section 2.1.9</td>
</tr>
<tr>
<td>units</td>
<td>OrgUnit[]</td>
<td>Organization</td>
<td>Section 2.2.2</td>
</tr>
<tr>
<td>updated</td>
<td>UTCDateTime</td>
<td>Card</td>
<td>Section 2.1.10</td>
</tr>
</tbody>
</table>

The following table lists the initial reserved usage entries of the "JSContact Properties" registry. All RFC section references are for [I-D.ietf-calext-jscontact]. The change controller for all these properties is IETF.

### Table 3: JSContact Properties with "reserved" usage

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Property Context</th>
<th>Reference or Description</th>
<th>Intended Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra</td>
<td>not applicable</td>
<td>not applicable</td>
<td>Section 1.5.2</td>
<td>reserved</td>
</tr>
</tbody>
</table>

The following table lists the initial reserved usage entries of the "JSContact Properties" registry. All RFC section references are for [I-D.ietf-calext-jscontact]. The change controller for all these properties is IETF.

### 3.6. Creation of the "JSContact Types" Registry

IANA is asked to create the "JSContact Types" registry. The purpose of this new registry is to avoid name collisions for JSContact type
names and provide a complete reference for all data types used for JSContact property values.

The registry entries sort alphabetically ascending by the "Type Name" column. Equal entries sort in any order.

The registry process for a new type is outlined in Section 3.3.

3.6.1. "JSContact Types" Registry Template

**Type Name:** the name of the type

**Reference or Description:** a brief description or RFC number and section reference where the Type is specified (may be omitted for "reserved" type names)

**Intended Usage:** common, reserved, or obsolete

**Since Version:** This defines the JSContact version on which this type definition is based on. The version **MUST** be one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

**Until Version:** This defines the JSContact version after which this type definition got obsoleted and **MUST NOT** be used in later versions. The Until Version value either **MUST** be not set, or one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

**Change Controller:** This is who may request a change to this entry's definition (IETF for RFCs from the IETF stream).

3.6.2. Initial Contents for the "JSContact Types" Registry

The following table lists the initial common usage entries of the JSContact Types registry. The Since Version for all types is 1.0. The Until Version for all types is not set. All RFC section references are for [I-D.ietf-calext-jscontact]. The change controller for all these properties is IETF.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>AddressComponent</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>Anniversary</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>Author</td>
<td>Section 2.8.3</td>
</tr>
<tr>
<td>Boolean</td>
<td>Section 1.3.2</td>
</tr>
<tr>
<td>Calendar</td>
<td>Section 2.4.1</td>
</tr>
<tr>
<td>Card</td>
<td>Section 2</td>
</tr>
<tr>
<td>CryptoKey</td>
<td>Section 2.6.1</td>
</tr>
</tbody>
</table>
Table 4: JSContact Types with "common" usage

The following table lists the initial reserved usage entries of the JSContact Types registry. All types are for version 1.0. All RFC section references are for [I-D.ietf-calext-jscontact]. The change controller for all these properties is IETF.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
<td>Section 2.6.2</td>
</tr>
<tr>
<td>EmailAddress</td>
<td>Section 2.3.1</td>
</tr>
<tr>
<td>Id</td>
<td>Section 1.4.1</td>
</tr>
<tr>
<td>Int</td>
<td>Section 1.4.2</td>
</tr>
<tr>
<td>LanguagePref</td>
<td>Section 2.3.4</td>
</tr>
<tr>
<td>Link</td>
<td>Section 2.6.3</td>
</tr>
<tr>
<td>Media</td>
<td>Section 2.6.4</td>
</tr>
<tr>
<td>Name</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td>NameComponent</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td>Nickname</td>
<td>Section 2.2.1.3</td>
</tr>
<tr>
<td>Note</td>
<td>Section 2.8.3</td>
</tr>
<tr>
<td>Number</td>
<td>Section 1.3.2</td>
</tr>
<tr>
<td>OnlineService</td>
<td>Section 2.3.2</td>
</tr>
<tr>
<td>Organization</td>
<td>Section 2.2.2</td>
</tr>
<tr>
<td>OrgUnit</td>
<td>Section 2.2.2</td>
</tr>
<tr>
<td>PartialDate</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>PatchObject</td>
<td>Section 1.4.3</td>
</tr>
<tr>
<td>PersonalInfo</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td>Phone</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>Pronouns</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>Relation</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>SchedulingAddress</td>
<td>Section 2.4.2</td>
</tr>
<tr>
<td>SpeakToAs</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>String</td>
<td>Section 1.3.2</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>Title</td>
<td>Section 2.2.4</td>
</tr>
<tr>
<td>UnsignedInt</td>
<td>Section 1.4.2</td>
</tr>
<tr>
<td>UTCDateTime</td>
<td>Section 1.4.5</td>
</tr>
</tbody>
</table>

3.7. Creation of the "JSContact Enum Values" Registry

IANA is asked to create the "JSContact Enum Values" registry. The purpose of the new registry is to allow interoperable extension of
semantics for JSContact properties with enumerable values. Each such property will have a subregistry of allowed values.

The registry entries sort alphabetically ascending by the "Property Name" column first, "Property Context" second, "Since Version" third. The enum values sort alphabetically ascending. Equal entries sort in any order.

The registry process for a new enum value or adding a new enumerable property is outlined in Section 3.3.

3.7.1. "JSContact Enum Values" Registry Property Template

This template is for adding a subregistry for a new enumerable property to the "JSContact Enum" registry.

Property Name: These are the name(s) of the property or properties where these values may be used. This MUST be registered in the "JSContact Properties" registry.

Context: This is the list of allowed object types where the property or properties may appear, as registered in the "JSContact Properties" registry. This disambiguates where there may be two distinct properties with the same name in different contexts.

Since Version: This defines the JSContact version on which this enum value definition is based on. The version MUST be one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

Until Version: This defines the JSContact version after which this enum value definition got obsoleted and MUST NOT be used in later versions. The Until Version value either MUST be not set, or one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

Change Controller: This is who may request a change to this entry's definition (IETF for RFCs from the IETF stream).

Initial Contents: This is the initial list of defined values for this enum, using the template defined in Section 3.7.2. A subregistry will be created with these values for this property name/context tuple.

3.7.2. "JSContact Enum Values" Registry Value Template

This template is for adding a new enum value to a subregistry in the JSContact Enum registry.
Enum Value: The verbatim value of the enum.

Reference or Description: A brief description or RFC number and section reference for the semantics of this value.

Since Version: The JSContact version on which the enum value definition is based on. The version MUST be one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

Until Version: The JSContact version after which this enum value got obsoleted and MUST NOT be used in later versions. The Until Version value either MUST be not set, or one of the allowed values of the version property in the JSContact Enum Value registry (see Table 1).

3.7.3. Initial Contents for the "JSContact Enum Values" Registry

For each subregistry created in this section, all RFC section references are for [I-D.ietf-calext-jscontact]. For all entries, the Since Version is 1.0, the Until Version is not set, the Change Controller is IETF.

Property Name: contexts
Context: Address
Initial Contents:

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billing</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>delivery</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>private</td>
<td>Section 1.5.1</td>
</tr>
<tr>
<td>work</td>
<td>Section 1.5.1</td>
</tr>
</tbody>
</table>

Table 6: JSContact Enum Values for contexts (Context: Address)

Property Name: contexts
Context: Calendar, CryptoKey, Directory, EmailAddress, LanguagePref, Link, Media, Nickname, OnlineService, Organization, Phone, Pronouns, SchedulingAddress
Initial Contents:

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>Section 1.5.1</td>
</tr>
<tr>
<td>work</td>
<td>Section 1.5.1</td>
</tr>
</tbody>
</table>

Table 7: JSContact Enum Values for contexts (Context: Calendar, CryptoKey, Directory, EmailAddress, LanguagePref, Link, Media, Nickname,
Property Name: features
Context: Phone
Initial Contents:

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fax</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>main-number</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>mobile</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>pager</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>text</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>textphone</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>video</td>
<td>Section 2.3.3</td>
</tr>
<tr>
<td>voice</td>
<td>Section 2.3.3</td>
</tr>
</tbody>
</table>

Table 8: JSContact Enum Values for features (Context: Phone)

Property Name: grammaticalGender
Context: SpeakToAs
Initial Contents:

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>animate</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>common</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>feminine</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>inanimate</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>masculine</td>
<td>Section 2.2.3</td>
</tr>
<tr>
<td>neuter</td>
<td>Section 2.2.3</td>
</tr>
</tbody>
</table>

Table 9: JSContact Enum Values for kind (Context: SpeakToAs)

Property Name: kind
Context: AddressComponent
Initial Contents:

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apartment</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>block</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>building</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>country</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>direction</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>district</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>floor</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>landmark</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>locality</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>name</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>number</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>postcode</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>Enum Value</td>
<td>Reference or Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>postOfficeBox</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>region</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>room</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>separator</td>
<td>Section 2.5.1</td>
</tr>
<tr>
<td>subdistrict</td>
<td>Section 2.5.1</td>
</tr>
</tbody>
</table>

Table 10: JSContact Enum Values for kind
(Context: AddressComponent)

Property Name: kind
Context: Anniversary
Initial Contents:
<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>birth</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>death</td>
<td>Section 2.8.1</td>
</tr>
<tr>
<td>wedding</td>
<td>Section 2.8.1</td>
</tr>
</tbody>
</table>

Table 11: JSContact Enum Values for kind (Context: Anniversary)

Property Name: kind
Context: Calendar
Initial Contents:
<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calendar</td>
<td>Section 2.4.1</td>
</tr>
<tr>
<td>freeBusy</td>
<td>Section 2.4.1</td>
</tr>
</tbody>
</table>

Table 12: JSContact Enum Values for kind (Context: Calendar)

Property Name: kind
Context: Card
Initial Contents:
<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>Section 2.1.4</td>
</tr>
<tr>
<td>device</td>
<td>Section 2.1.4</td>
</tr>
<tr>
<td>group</td>
<td>Section 2.1.4</td>
</tr>
<tr>
<td>individual</td>
<td>Section 2.1.4</td>
</tr>
<tr>
<td>location</td>
<td>Section 2.1.4</td>
</tr>
<tr>
<td>org</td>
<td>Section 2.1.4</td>
</tr>
</tbody>
</table>

Table 13: JSContact Enum Values for kind (Context: Card)

Property Name: kind
Context: Directory
Initial Contents:
<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Section 2.6.2</td>
</tr>
<tr>
<td>Property Name: kind</td>
<td>Context: Directory</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Initial Contents:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kind</th>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>entry</td>
<td>Section 2.6.2</td>
</tr>
</tbody>
</table>

Table 14: JSContact Enum Values for kind (Context: Directory)

<table>
<thead>
<tr>
<th>Property Name: kind</th>
<th>Context: Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Contents:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kind</th>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>contact</td>
<td>Section 2.6.3</td>
</tr>
</tbody>
</table>

Table 15: JSContact Enum Values for kind (Context: Link)

<table>
<thead>
<tr>
<th>Property Name: kind</th>
<th>Context: Media</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Contents:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kind</th>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>logo</td>
<td>Section 2.6.4</td>
</tr>
<tr>
<td></td>
<td>photo</td>
<td>Section 2.6.4</td>
</tr>
<tr>
<td></td>
<td>sound</td>
<td>Section 2.6.4</td>
</tr>
</tbody>
</table>

Table 16: JSContact Enum Values for kind (Context: Media)

<table>
<thead>
<tr>
<th>Property Name: kind</th>
<th>Context: NameComponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Contents:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kind</th>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>credential</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>generation</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>given</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>given2</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>separator</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>surname</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>surname2</td>
<td>Section 2.2.1</td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>Section 2.2.1</td>
</tr>
</tbody>
</table>

Table 17: JSContact Enum Values for kind (Context: NameComponent)

<table>
<thead>
<tr>
<th>Property Name: kind</th>
<th>Context: PersonalInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Contents:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kind</th>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>expertise</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td></td>
<td>hobby</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td></td>
<td>interest</td>
<td>Section 2.8.4</td>
</tr>
</tbody>
</table>
### Property Name: kind

**Context:** Title  

**Initial Contents:**

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>Section 2.2.4</td>
</tr>
<tr>
<td>title</td>
<td>Section 2.2.4</td>
</tr>
</tbody>
</table>

### Property Name: level

**Context:** PersonalInfo  

**Initial Contents:**

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td>low</td>
<td>Section 2.8.4</td>
</tr>
<tr>
<td>medium</td>
<td>Section 2.8.4</td>
</tr>
</tbody>
</table>

### Property Name: relation

**Context:** Relation  

**Initial Contents:**

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Reference or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquaintance</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>agent</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>child</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>colleague</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>contact</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>co-resident</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>co-worker</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>crush</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>date</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>emergency</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>friend</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>kin</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>me</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>met</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>muse</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>neighbor</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>parent</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>sibling</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>spouse</td>
<td>Section 2.1.8</td>
</tr>
<tr>
<td>sweetheart</td>
<td>Section 2.1.8</td>
</tr>
</tbody>
</table>
Table 21: JSContact Enum Values for relation (Context: Relation)

<table>
<thead>
<tr>
<th>Property Name: phoneticSystem</th>
<th>Context: Address, Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enum Value</strong></td>
<td><strong>Reference or Description</strong></td>
</tr>
<tr>
<td>ipa</td>
<td>Section 1.5.5</td>
</tr>
<tr>
<td>jyut</td>
<td>Section 1.5.5</td>
</tr>
<tr>
<td>piny</td>
<td>Section 1.5.5</td>
</tr>
</tbody>
</table>

Table 22: JSContact Enum Values for phoneticSystem (Context: Address, Name)

4. Security Considerations

Contact information is very privacy-sensitive. It can reveal the identity, location and credentials information, employment status, interests and hobbies, and social network of a user. Its transmission and storage must be done carefully to protect it from possible threats, such as eavesdropping, replay, message insertion, deletion, modification, and on-path attacks.

The data being stored and transmitted may be used in systems with real-world consequences. For example, a malicious actor might provide JSContact data that uses the name of another person but insert their contact details to impersonate the unknown victim. Such systems must be careful to authenticate all data they receive to prevent them from being subverted and ensure the change comes from an authorized entity.

This document only defines the data format; such considerations are primarily the concern of the API or method of storage and transmission of such files.

4.1. JSON Parsing

The security considerations of [RFC8259] apply to the use of JSON as the data interchange format.

As for any serialization format, parsers need to thoroughly check the syntax of the supplied data. JSON uses opening and closing brackets for several types and structures, and it is possible that the end of the supplied data will be reached when scanning for a matching closing bracket; this is an error condition, and implementations need to stop scanning at the end of the supplied data.
JSON also uses a string encoding with some escape sequences to encode special characters within a string. Care is needed when processing these escape sequences to ensure that they are fully formed before the special processing is triggered, with special care taken when the escape sequences appear adjacent to other (non-escaped) special characters or adjacent to the end of data (as in the previous paragraph).

If parsing JSON into a non-textual structured data format, implementations may need to allocate storage to hold JSON string elements. Since JSON does not use explicit string lengths, the risk of denial of service due to resource exhaustion is small, but implementations may still wish to place limits on the size of allocations they are willing to make in any given context, to avoid untrusted data causing excessive memory allocation.

4.2. URI Values

Several JSContact properties contain URIs as values, and processing these properties requires extra care. Section 7 of [RFC3986] discusses security risks related to URIs.

Fetching remote resources carries inherent risks. Connections must only be allowed on well-known ports, using allowed protocols (generally, just HTTP/HTTPS on their default ports). The URL must be resolved externally and not allowed to access internal resources. Connecting to an external source reveals IP (and therefore often location) information.

A maliciously constructed JSContact object may contain a very large number of URIs. In the case of published address books with a large number of subscribers, such objects could be widely distributed. Implementations should be careful to limit the automatic fetching of linked resources to reduce the risk of this being an amplification vector for a denial-of-service attack.

5. References

5.1. Normative References


5.2. Informative References


Authors' Addresses

Robert Stepanek
Fastmail
PO Box 234, Collins St West
Melbourne VIC 8007
Australia

Email: rsto@fastmailteam.com

Mario Loffredo
IIT-CNR
Via Moruzzi,1
56124 Pisa
Italy

Email: mario.loffredo@iit.cnr.it